Personal SE

Arrays
Pointers
Strings
Array Identifiers & Pointers

char message[] = "Hello" ;
Array Identifiers & Pointers

char message[] = “Hello” ;

message

Question: So what exactly is message?
Array Identifiers & Pointers

char message[] = “Hello” ;

message
Hello \0

Question: So what exactly is message?

Answer: In C, an array name is a constant pointer that references the 0th element of the array's storage.
Array Identifiers & Pointers

char message[] = “Hello” ;

Question: So what exactly is message?

Answer: In C, an array name is a constant pointer that references the 0th element of the array's storage.

Constant means it cannot be changed (just as we can't change the constant 3).
Consequences - Part 1

```c
char message[] = "Hello";
```

What is `*message`?
Consequences - Part 1

char message[] = "Hello" ;

What is *message?

*message == 'H'
Consequences - Part 1

char message[] = “Hello” ;

What is \*message?

\*message == 'H'

What is another expression for message?

```
char message[] = "Hello";

\*message == 'H'
```
Consequences - Part 1

cchar message[] = “Hello” ;

What is *message?

*message == 'H'

What is another expression for message?

message == &message[0]
Consequences - Part 1

```
char message[] = "Hello";
```

What is *message?

```
*message == 'H'
```

What is another expression for message?

```
message == &message[0]
```

What is another expression for message[4]?
Consequences - Part 1

char message[] = “Hello”;

What is \*message?

\*message == 'H'

What is another expression for message?

message == &message[0]

What is another expression for message[4]?


That's right - we can add or subtract an integer and a pointer to get a pointer to the element a certain distance from the original!
char *hi = "Hello";
Creates a constant string "Hello" and initializes the hi pointer to point to the 'H' (the initial character).
char *hi = "Hello";
Creates a constant string "Hello" and initializes the hi pointer to point to the 'H' (the initial character).

char message[] = "Greetings!";
Allocates space for the array message and initializes its contents to the string "Greetings!".
char *hi = “Hello” ;
Creates a constant string “Hello” and initializes the hi pointer to point to the 'H' (the initial character).

char message[] = “Greetings!” ;
Allocates space for the array message and initializes its contents to the string “Greetings!”.

char *p_mesg = message ;
Initializes p_mesg to point to the initial element of message.
char *hi = “Hello” ;
Creates a constant string “Hello” and initializes the hi pointer to point to the 'H' (the initial character).

char message[] = “Greetings!” ;
Allocates space for the array message and initializes its contents to the string “Greetings!”.

char *p_mesg = message ;
Initializes p_mesg to point to the initial element of message.

char ch ;
p_mesg++;  
ch = *p_mesg ;
Declares ch, advances p_mesg by one element, and sets ch to the character p_mesg points to (in this case 'r').
char *hi = "Hello";
char ch0 = hi[1]; // ch = 'e'

Pointers can be indexed
char *hi = "Hello";
char ch0 = hi[1];  // ch = 'e'

Pointers can be indexed

char *hp = hi;       // initially the same as hi
char ch1;
ch1 = *hp++;

Post-increment: \texttt{ch1 = *hp then hp += 1 (ch1 == 'H' \ and \ hp == hi + 1)}
char *hi = “Hello”;
char ch0 = hi[1] ;  // ch = 'e'

Pointers can be indexed

char *hp = hi ;    // initially the same as hi
char ch1 ;
ch1 = *hp++ ;
Post-increment: ch1 = *p then p += 1   (ch1 == 'H' and p == hi + 1)

char ch2 ;
ch2 = *++hp ;
Pre-increment: hp += 1 then ch1 = *hp (hp == hi + 2 and ch2 == 'l')
Alternatives & Idioms - 1

char *hi = “Hello”; 
char ch0 = hi[1];  // ch = 'e'

Pointers can be indexed

char *hp = hi; // initially the same as hi
char ch1;
ch1 = *hp++;
Post-increment: ch1 = *p then p += 1 (ch1 == 'H' and p == hi + 1)

char ch2;
ch2 = *++hp;
Pre-increment: hp += 1 then ch1 = *hp (hp == hi + 2 and ch2 == 'l')

Also have pre and post decrement with --
char *p;
char ch;
while( ch = *p++ ) {
    // process characters until end of string.
}
char *p;
char ch;
while( ch = *p++ ) {
    // process characters until end of string.
}

if( *p ) {  // true if p points to a “real” character
    if( *p != '\0') {   // easier to read
char *p;
char ch;
while( ch = *p++) {
    // process characters until end of string.
}

if( *p ) {  // true if p points to a “real” character
if( *p != '\0') {  // easier to read

if( !*p ) { // true if p points to a NUL character.
if( *p == '\0' ) {  // easier to read